3.7 Variable Assignment

Recall instance variables in Python:

```
class BankAccount:

  def __init__(self):
    self.balance = 0 # instance var

  def deposit(self, amount):
    self.balance += amount
    return self.balance

  def withdraw(self, amount):
    self.balance -= amount
    return self.balance
```

Want to allow procedures to “share” data this way:

```
--> let bal = 0
    in letrec
         deposit(amt) =
           let d = set bal = +(bal, amt)
           in bal
         withdraw(amt) =
           let d = set bal = -(bal, amt)
           in bal
       in (deposit 100)
```

- Add to grammar:

<expression> ::= **set** <identifier> **=** <expression>

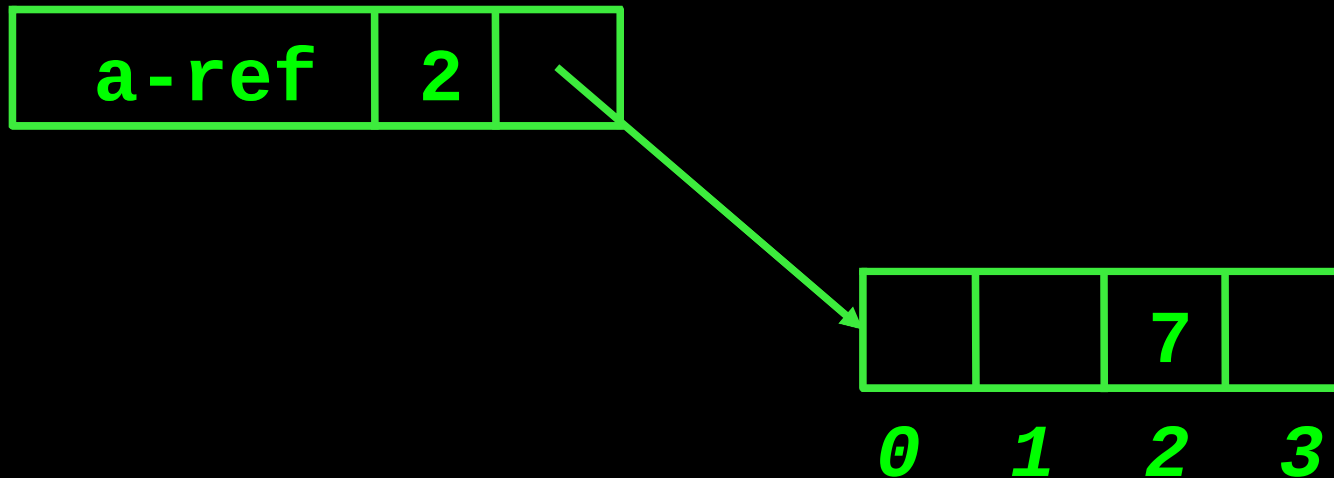
```
varassign-exp (id rhs-exp)
```

- Add to **eval-expression**: Must *alter* current env, not just add to it.

- So need a new datatype: **reference**, with operations **setref** and **deref**

Use Scheme *vector* (more efficient than list):

```
(define-datatype reference reference?
    (a-ref ; just one variant
      (position integer?)
      (vec vector?)))
```



Now we can define **setref** and **deref:**

```
(define setref!
  (lambda (ref val)
    (cases reference ref
      (a-ref (pos vec)  ; only variant
        (vector-set! vec pos val)))))

(define deref
  (lambda (ref)
    (cases reference ref
      (a-ref (pos vec)  ; only variant
        (vector-ref vec pos)))))
```

Use **setref** in **eval-expression**:

```
(define eval-expression
    (lambda (exp env)
      (cases expression exp
        ...
        (varassign-exp (id rhs-exp)
          (begin
            (setref!
              (apply-env-ref env id)
              (eval-expression rhs-exp env))
            1)) ; huh?
```

So what does **apply-env-ref** do?

```
(define apply-env-ref
  (lambda (env sym)
    (cases environment env
      (empty-env-record ()
        (eopl:error 'apply-env-ref “No binding...”))
      (extended-env-record (syms vals env)
        (let ((pos (list-find-position sym syms)))
          (if (number? pos) ; found
              (a-ref pos vals)
              (apply-env-ref env sym)))))))
```

- Can redefine **apply-env** in terms of **apply-env-ref:**

```
(define apply-env
  (lambda (env sym)
    (deref (apply-env-ref env sym))))
```

- Why **deref** ?

- Because **apply-env-ref** creates a **reference** structure.

- Recall original example:

```
--> let bal = 0
   in letrec
         deposit(amt) =
           let d = set bal = +(bal, amt)
           in bal
         withdraw(amt) =
           let d = set bal = -(bal, amt)
           in bal
      in (deposit 100)
```

- Why is this uninteresting? What do we need?